Parallel Seam Carving: Performance Analysis

Erik Pahor, Jaka Škerjanc

I. Introduction

Seam carving is a content-aware image resizing algorithm that removes the least important seams (connected paths of pixels) from an image. In this project, we parallelized the seam carving algorithm using OpenMP to improve its performance on multi-core systems. We measured the execution time for different image sizes and core configurations, and computed the speed-up achieved by the parallel implementation. This report presents the results of our experiments and analyzes the performance of the parallelized algorithm.

II. Experiments

A. Experimental Setup

We tested the parallelized seam carving algorithm on five different image sizes: 592x480, 896x768, 1892x1200, 3712x2160, and 7552x4320. For each image size, we ran the algorithm multiple times, removing 128 seams each time, and averaged the execution time to obtain representative results. We experimented with different numbers of cores and threads, including configurations where the number of threads exceeded the number of cores, to evaluate the impact of hyper-threading.

B. Sequential vs. Parallel Performance

Table I shows the average execution time for the sequential implementation of the seam carving algorithm using two different energy computation methods: `col_diff_grad` and `sobel`. As expected, the execution time increases with the image size, with the largest image (7552x4320) taking over 112 seconds for `col_diff_grad` and 312 seconds for `sobel`.

Table I
Average execution time (in seconds) for the sequential algorithm.

Energy Method	Image Size	Average Time (sec)
sobel	592x480	3.60515
sobel	896x768	7.86012
sobel	1892x1200	5.35351
sobel	3712x2160	77.639
sobel	7552x4320	312.353
col_diff_grad	592x480	**1.43286**
col_diff_grad	896x768	**3.35978**
col_diff_grad	1892x1200	**2.15124**
col_diff_grad	3712x2160	**28.1146**
col_diff_grad	7552x4320	**112.177**

C. Parallel Performance

Table II presents the average execution time for the parallel implementation with different core and thread configurations. For smaller images (592x480), the best performance was achieved with two cores and two threads, suggesting that the overhead of parallelization outweighs the benefits for these sizes. However, for larger images (896x768, 1892x1200, 3712x2160, and 7552x4320), the parallel implementation significantly reduces the execution time, with the best performance achieved using 8 cores and threads for medium sized images and 16 cores and threads or more for large images.

Table II
Average execution time (in seconds) for the parallel algorithm with different core and thread configurations.

Cores	Threads	Image Size	Average Time (sec)
2	1	592x480	0.7177
2	1	896x768	1.2982
2	1	1892x1200	0.7734
2	1	3712x2160	11.1835
2	1	7552x4320	47.9936
2	2	592x480	**0.6307**
2	2	896x768	1.1683
2	2	1892x1200	0.9642
2	2	3712x2160	10.4110
2	2	7552x4320	43.7794
4	4	592x480	0.7634
4	4	896x768	1.3829
4	4	1892x1200	0.7832
4	4	3712x2160	6.4311
4	4	7552x4320	25.1645
8	8	592x480	0.7211
8	8	896x768	**1.1608**
8	8	1892x1200	**0.6170**
8	8	3712x2160	4.4833
8	8	7552x4320	16.4680
16	16	592x480	0.9811
16	16	896x768	1.4717
16	16	1892x1200	0.6989
16	16	3712x2160	**3.8650**
16	16	7552x4320	**14.8920**

D. Speed-Up Analysis

The speed-up $S = t_s/t_p$ was computed for each image size, where t_s is the sequential execution time and t_p is the parallel execution time with the optimal core/thread configuration. Figure 1 shows the speed-up achieved for each image size. The largest speed-up was observed for the 7552x4320 image, where the parallel implementation achieved a speed-up by more than 7.5 times compared to the sequential version.

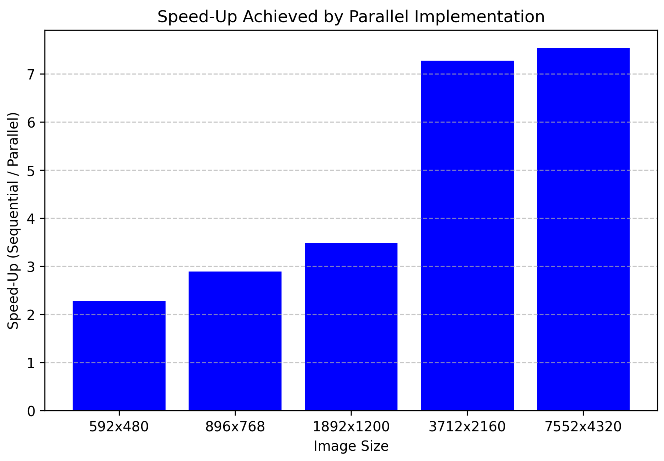

Figure 1. Speed-up achieved by the parallel implementation for different image sizes.

E. *Failure Cases and Improvements*

For smaller images, the parallel implementation did not provide a very large speed-up due to the overhead of parallelization. Additionally, for very large images (7552x4320), the performance seemed to improve the more threads were used and did not hit the bottleneck. Future work could focus on optimizing the parallel algorithm for smaller images and reducing the overhead for large images by improving load balancing.

III. Experiments (Continued)

A. *Triangular Implementation Results*

After addressing memory management issues in smaller images, we implemented a triangular-blocked approach for cumulative energy calculation. Table III shows the performance of this optimized implementation:

Table III
Average execution times (seconds) for triangular implementation

Cores	Threads	Image Size	Avg Time (sec)
2	1	896x768	0.1866
2	1	1892x1200	0.1236
2	1	3712x2160	**1.0222**
2	1	7552x4320	3.8041
2	2	896x768	0.1927
2	2	1892x1200	0.1303
2	2	3712x2160	1.3618
2	2	7552x4320	3.2807
4	4	896x768	**0.1726**
4	4	1892x1200	**0.1122**
4	4	3712x2160	1.3045
4	4	7552x4320	2.9371
8	8	896x768	0.2020
8	8	1892x1200	0.1183
8	8	3712x2160	1.2956
8	8	7552x4320	**2.9152**
16	16	896x768	0.2533
16	16	1892x1200	0.1343
16	16	3712x2160	1.3625
16	16	7552x4320	3.0192

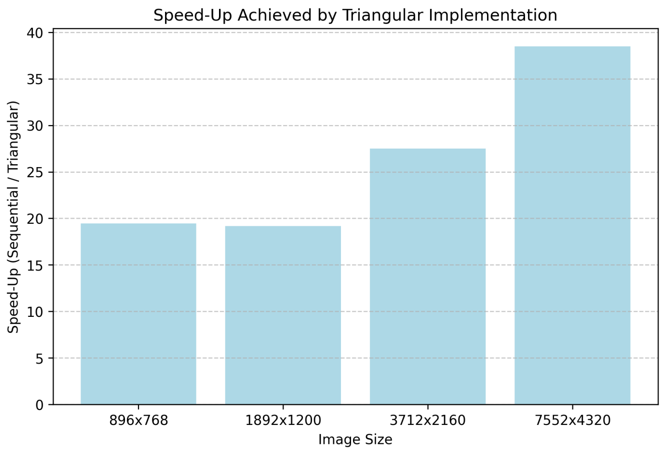

Figure 2. Speed-up achieved by triangular implementation (vs sequential)

B. *Enhanced Speed-Up Analysis*

The triangular implementation shows significantly better scaling characteristics compared to the basic parallel approach. Figure 2 demonstrates remarkable speed-up factors, particularly for large images:

IV. Conclusion

The parallelized seam carving algorithm achieved significant speed-up for large images, with the best performance observed using up to 8 cores, than the gains seem to be very small. However, for smaller images, the overhead of parallelization outweighed the benefits. Hyper-threading provided limited improvements and often degraded performance due to resource contention. Future work could focus on optimizing the algorithm for smaller images and further reducing the overhead for large images.